НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА
на здобуття ступеня бакалавра

з напряму підготовки
6.050101 “Комп’ютерні науки’’

на тему: Інструментальні засоби міграції даних із реляційної бази даних до онтології .owl формату

Виконав: студент 4 курсу, групи ТР-51

Талах Олексій Миколайович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач Дацюк Оксана Антонівна _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “Комп’ютерні науки’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Талаху Олексію Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “ Інструментальні засоби міграції даних із реляційної бази даних до онтології .owl формату ”
керівник роботи старший викладач Дацюк Оксана Антонівна
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Mac OS, мова програмування Java, мова програмування JavaScript.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації, розробити програмне забезпечення, розробити користувацький інтерфейс,
5. Перелік ілюстраційного матеріалу архітектура проекту, графічне представлення інтерфейсу, приклади роботи програмного модулю.
6. Публікації: Міжнародна наукова інтернет-конференція “Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення”, м. Тернопіль, 7 травня 2019 року

Дата видачі завдання ”___”_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	14.10.2018-23.12.2018	
2.	Розробка архітектури та загальної структури системи	2.02.2019-3.03.2019	
3.	Розробка структур окремих підсистем	4.03.2019-14.04.2019	
4.	Підготовка матеріалів	15.04.2019-18.04.2019	
5.	Програмна реалізація системи	18.04.2019-14.05.2019	
6.	Захист програмного продукту	15.05.2019	
7.	Оформлення пояснювальної записки	16.05.2019-3.06.2019	
8.	Передзахист	28.05.2019	
9.	Захист	17.06.2019-22.06.2019	

Студент _____ Талах О.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Дацюк О. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Обсяг дипломної роботи складає 61 сторінок, 20 рисунків, 4 додатки та 10 посилань.

Мета роботи — створити програмний додаток, який дозволяє перенести дані із реляційної бази даних до онтології .owl формату.

У ході роботи було розглянуто редактор онтологій Protégé та плагіна Ontop Проаналізовано підходи до конвертації типів даних з онтології в реляційну базу даних та їх недоліки. Розроблено систему перенесення даних з реляційної бази даних до онтології.

Результати дослідження доповідалися на одній науковій конференції.

Ключові слова: онтологія, Protégé, OWL, RDF, Ontop, individuals.

ANNOTATION

Volume of the thesis is 61 pages, 20 figures, 4 annexes and 10 references.

The purpose of the work is to create a software application that allows you to transfer data from the relational database to the .owl format ontology.

In the course of work, the ontology editor Protégé and Ontop plug-in Ontop were analyzed. Approaches to converting types of data from ontology into a relational database and their disadvantages were analyzed. A system for transferring data from a relational database to an ontology has been developed.

The results of the study were reported at one scientific conference.

Key words: ontology, Protégé, OWL, RDF, Ontop, individuals.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IDE — Integrated Development Environment

XML — Extensible Markup Language

UML — Unified Modeling Language

OBDB — Ontologies Based Databases

OWL — Web Ontology Language

RDF — Resource Description Framework

RDFS — Resource Description Framework Schema

SQL — Structured Query Language

SPARQ — is an RDF query language

JDK — Java Development Kit

GPL — General Public License

JVM — Java Virtual Machine

MVC — Model View Controller

CLI — Command Line Interface

JDBC — Java DataBase Connectivity

JPA — Java Persistence API

JMS — Java Message Service

RxJS — Reactive Extensions For JavaScript

ORM — Object-Relational Mapping

URI — Uniform Resource Identifier

HTML — HyperText Markup Language

CSS — Cascading Style Sheets

W3C — World Wide Web Consortium

REST — Representational State Transfer

СКБД — Система керування базами даних

ВСТУП

Онтологію застосовують як формалізоване представлення знать про певну предметну область, придатну для автоматизованої обробки. Її завжди супроводжує деяка концепція цієї області інтересів. Найчастіше ця концепція виражається за допомогою визначення базових об’єктів (індивідуумів, атрибутів, процесів) і відношень між ними [1].

Переваги онтології полягають в поліпшенні взаємодії розробників та програмних агентів, уніфікації обміну даних, формалізації процесів специфікації, підвищення надійності та забезпеченні багаторазовості використання. Онтології зазвичай містять класи, екземпляри цих класів, їхні атрибути та значення цих властивостей, а також відношення між класами та екземплярами класів. Крім того, онтологія може містити певні обмеження на використання класів та їх відношень [2]. Через це робота з онтологією напряму потребує певних навичок, а для заповнення даними потрібно використати велику кількість часу. Так в онтологію досить незручно заносити дані. Та й працювати з великими об’ємами даних в онтології незручно. Для цього більш пристосовані реляційні бази даних (РБД). Реляційні БД сьогодні мають досить потужні механізми БД призначені для організації коректного зберігання та швидкої обробки великих об’ємів даних. Можливість заповнення онтології з бази даних на основі співвідношення стовпців таблиць бази даних та атрибутів онтології спростить та автоматизує роботу користувача.

Програмний додаток розроблявся з використанням мов програмування Java, JavaScript та фреймворків Angular 7, Spring та Hibernate з використанням бібліотеки eduStandfordProtege для аналізу файлу онтології.

У завдання дипломного проекту входять:

1. Ознайомитися із редактором Protege та виявити основні недоліки в користуванні;
2. Створити ієрархію онтології;
3. Розробити конвертацію типів даних із реляційної бази даних до онтології;
4. Розробити програмний продукт міграції даних.

1 ЗАДАЧА РОЗРОБКИ ІНСТРУМЕНТАЛЬНОГО ЗАСОБУ МІГРАЦІЇ ДАНИХ ІЗ РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ ДО ОНТОЛОГІЇ .OWL ФОРМАТУ

Метою розробки є створення програмного додатку, який дозволяє перенести дані із реляційної бази даних до онтології .owl формату. Автоматизувати процес заповнення даними онтології для зменшення використаного часу оператора.

Призначенням даного програмного засобу є спосіб швидкого наповнення онтології екземплярами класів використовуючи дані з реляційної бази даних.

Програмний засіб розробляється для комп'ютерів з будь-якою операційною системою, так як це WEB-додаток використати його може кожний в кого є браузер.

Вхідні дані: файл онтології з розширенням *.owl, параметри для підключення реляційної бази даних, SQL-запит в реляційну базу даних.

Вихідні дані: файл онтології з розширенням *.owl.

Необхідними можливостями, які має забезпечувати модуль, є:

- завантаження файлу онтології з комп'ютера користувача;
- аналіз файлу онтології;
- підключення до реляційної бази даних;
- валідація підключення до реляційної бази даних;
- написання SQL-запиту для отримання даних з реляційної бази даних;
- валідація коректності SQL-запиту;
- запис екземплярів класів в онтологію;
- завантаження оновленого файлу онтології.

2 АНАЛІЗ ПРОБЛЕМИ МІГРАЦІЇ ДАНИХ

2.1 Опис предметної області

Онтології зазвичай містять такі елементи: класи, їхні екземпляри. Властивості та значення цих властивостей, а також відношення між класами та екземплярами. Крім того, онтологія може містити певні обмеження на використання класів та їх відношень [3].

Екземпляри — це основні, компоненти онтології. Екземпляри можуть являти собою як фізичні об'єкти (люди, будинки, планети), так і абстрактні (числа, слова). Онтологія може обійтися й без конкретних об'єктів. Однак, однією з головних цілей онтології є класифікація таких об'єктів, тому вони також включаються.

Класи — абстрактні групи, колекції або набори об'єктів. Вони можуть містити в собі екземпляри, інші класи, або ж сполучення обох.

Об'єкти в онтології можуть мати властивості. Кожна властивість має хоча б ім'я й значення, і використовується для зберігання інформації.

Значення атрибута може бути складеним типом даних.

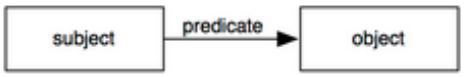
Важлива роль атрибутів розкривається в тому, щоб визначати відношення між об'єктами онтології. Зазвичай відношенням є властивість, значенням якого є інший об'єкт.

Для представлення онтології в пам’яті комп’ютера використовують спеціальні мови опису: RDF, OWL та інші. Для роботи з мовами опису онтологій існують редактори онтологій, які дозволяють створювати та змінювати структуру онтології, наповнювати її даними та можуть мати графічний редактор для відображення онтології.

OWL — це стандарт W3C, мова для семантичних тверджень, розроблена як розширення RDF і RDFS.

RDF — це універсальний засіб поділу будь-яких знань на частини. Завдяки ньому задаються відповідні правила стосовно змісту знань. Ідея полягає в легкому описанні будь-якого факту у структурованій формі таким чином, щоб його можна було обробити комп'ютерною програмою.

Факт являє собою триплет виду: (subject, predicate, object) (Рисунок 2.1).

Graphical form	
Triple	`subject predicate object`
Relational form	`predicate(subject,object)`

RDF/XML	`<rdf:Description rdf:about="subject">`
	`  <ex:predicate>`
	`  <rdf:Description rdf:about="object"/>`
	`  </ex:predicate>`
	`</rdf:Description>`

Рисунок 2.1 – Синтаксис триплету RDF

Subject, predicate, object — це назви сутностей реального світу, конкретних або абстрактних. Крім цього object може бути текстовою строкою — "літералом" [4].

Існують два, пов'язаних між собою, способи представлення інформації, яка закодована в RDF. Перший спосіб — зчитувати її списком тверджень (триплетів). Другий спосіб — зчитувати її графом.

Граф складається з вузлів, з'єднаних ребрами. У RDF вузли — це імена (але не самі сутності), а ребра — твердження.

Кожна стрілка (ребро) — це RDF-твердження: ім'я на початку стрілки — це subject твердження, ім'я на кінці стрілки — object, і ім'я у самій стрілки — predicate (Рисунок 2.2).

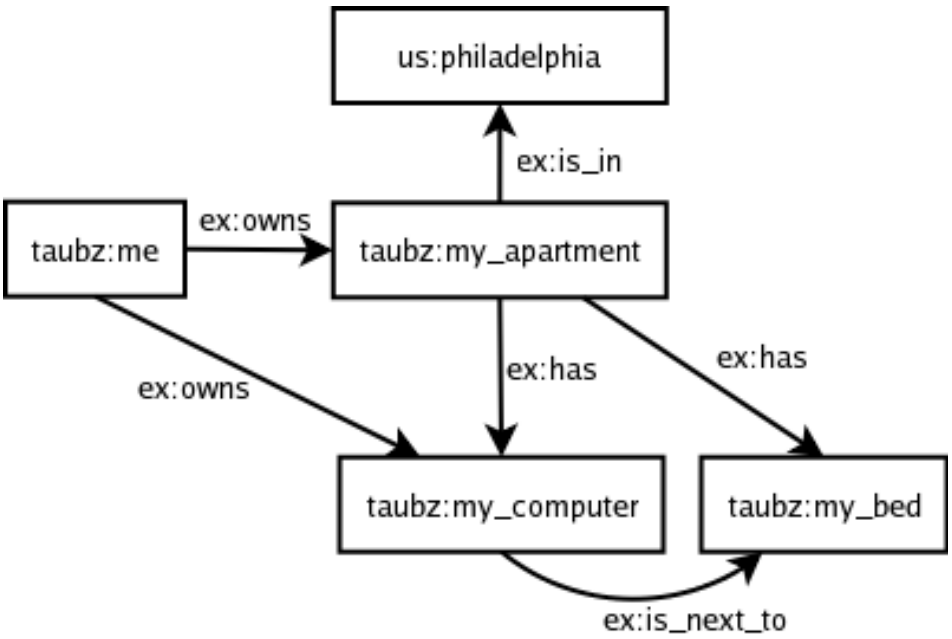


Рисунок 2.2 – Представлення онтології у вигляді графу

Коли RDF представлений у вигляді графа, він містить всю ту ж інформацію, що і виписаний у вигляді триплетів, але графічне представлення дозволяє легше побачити структуру даних [5].

2.2 Існуючі рішення

Protégé - безкоштовний редактор онтологій з відкритим вихідним кодом і системою керування знаннями. Protégé надає графічний інтерфейс користувача для визначення онтологій та внесення екземплярів класів для наповнення даними онтології. Protégé також включає дедуктивні класифікатори для підтвердження послідовності моделей та виведення нової інформації на основі аналізу онтології. (Рисунок 2.3)

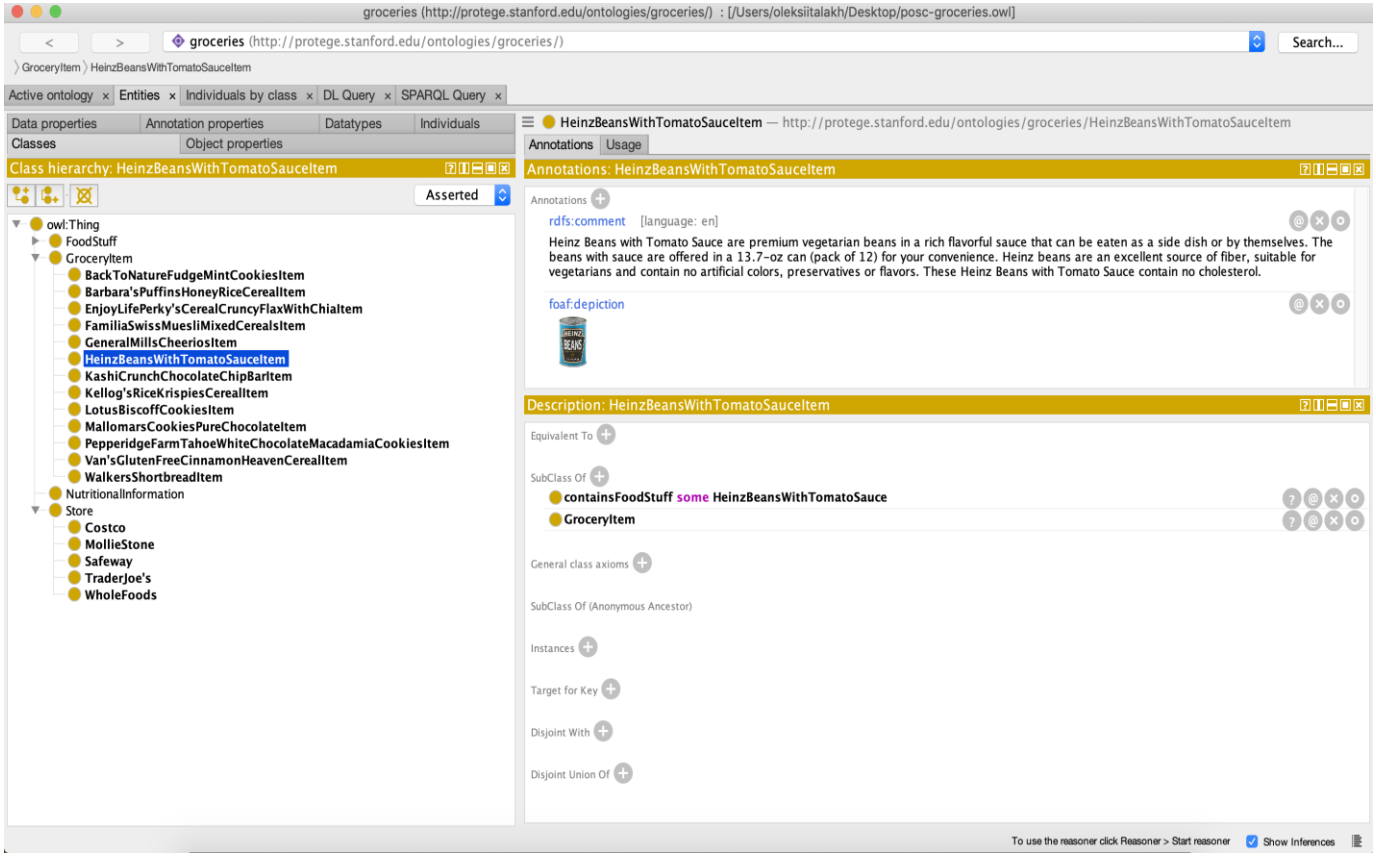


Рисунок 2.3 – Інтерфейс програми Protégé

Зв'язок між БД та онтологією може бути встановлений, якщо інформація, представлена в онтології відповідає даним БД. При цьому організація комплексної роботи онтології з базою даних проводиться за певними правилами. Є декілька підходів для сумісного використання даних онтологією і БД.

Перший підхід ґрунтується на використанні однієї концептуальної моделі для БД і онтології. В [6] запропоновано використовувати UML-діаграму для генерації схеми онтології та БД.

Другий підхід орієнтований на створенні схем баз даних з онтологій. Звичайно, в цьому випадку при записі великих об’ємів даних може бути втрачено семантичний зв’язок між даними.

Протилежним підходом є отримання онтології з реляційної бази даних. Це найбільш поширений метод, але така онтологія має обмежений набір зв’язків і залежностей між класами, що легко компенсується ручним коригуванням структури онтології [7].

Одним з різновидів першого підходу є використання БД на основі онтологій OBDB (Ontologies Based Databases). Де реляційна БД використовується для зберігання даних, представлених в онтології. В даному випадку схема БД не завжди співпадає зі схемою онтології, але робота підтримується за допомогою окремих спеціально створених комунікативних зв'язків. Такі зв'язки часто приходиться встановлювати вручну.

Ще однією альтернативною сумісного використання БД та онтології є отнологія, яка описує структуру концептуалізація моделі реляційної бази даних.

Одним із рішень роботи онтології з базою даних є плагін Ontop розроблений спеціально для Protégé. Основною роботою плагіна є підключення до реляційної бази даних та написанню SPARQL-відображень, на основі яких можна робити запити до існуючої реляційної бази даних

Однак існуючий функціонал має свої недоліки:

— для виконання кожного SQL-запиту потрібно написати власний SPARQL-mapping для цього запиту;

— при кожному відкритті Protégé прийдеться знову встановлювати підключення до реляційної бази даних;

— підключення плагіну не завжди завершується успіхом та призводить до відмови роботи самого редактора Protégé.

Іншим способом міграції даних для їх використання є перенесення їх вручну але він має низку недоліків:

— етапи для перенесення даних вимагають від користувача постійної концентрації та уваги;

— дані необхідно вводити в поля, які розташовані на різних формах, доступ до яких здійснюється натиском курсору миші по програмним кнопкам головної форми Protégé;

— введення великої кількості даних до онтології займає багато часу.

2.3 Висновки до розділу

У даному розділі описано складові елементи онтології.

Розглянуто основні підходи до сумісної роботи з даними онтології та реляційної БД. Проаналізовані функціональні можливості редактору Protégé та плагіну Ontop для роботи з реляційною базою даних, та можливості перенесення даних вручну. Виявлені їхні основні недоліки. Постійне повторне налаштування підключення до реляційної бази даних, написання додаткових перевірок за допомогою SPARQL-запитів та найважливішим це багато часу та зусиль для інсталяції даного плагіна. Недоліком заповнення даних вручну є те що додавання навіть одного нового елементу вимагає від користувача значної кількості етапів, створення низки елементів для оператора не приносить ніякого задоволення і, навіть, навпаки вимагає багато часу.

3 ЗАСОБИ РОЗРОБКИ

У процесі створення програмного продукту важливо правильно обрати засоби реалізації. Проаналізувавши основні помилки при створенні програмного забезпечення і підкресливши важливі фактори для створення власного додатку, враховувалися різні фактори. Основним завданням була економія часу на розробку, якість, надійність і швидкість роботи програмного продукту.

Система складається з двох частин, а саме серверної та клієнтської частини. Приклад такої архітектури можна розглянути на рисунку 3.1.

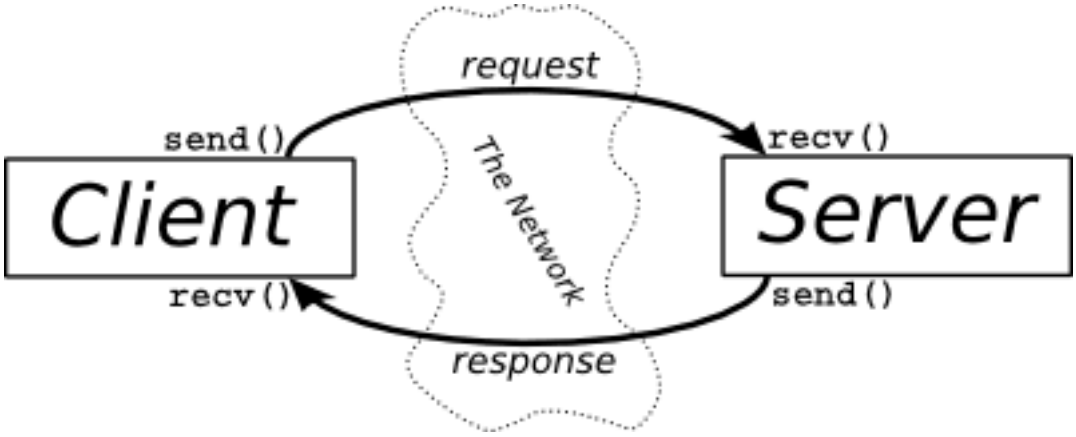


Рисунок 3.1 – Клієнт-серверна архітектура

Аналізуючи можливі підходи для досягнення поставленої мети було вибрано два найпопулярніші стеки технологій для розробки серверної та клієнтської частини програми. Для серверної частини було обрано мову програмування Java з фреймворками Spring та Hibernate а для клієнтської мову програмування JavaScript з фреймворком Angular 7 та додатковими бібліотеками. Середовищем розробки було обрано IntelliJ IDEA та WebStorm відповідно.

3.1 Мова програмування Java

Java — це об’єктно-орієнтована мова програмування яка була розроблена компанією Sun Microsystems яку далі викупила компанія Oracle. На сьогодні проект належить OpenSource і розповсюджується по ліцензії GPL. В OpenJDK вносять свій вклад різні великі компанії такі як Google, IBM, RedHat, Oracle, JetBrains. Також на основі даної OpenJDK вони розроблюють власні збірки JDK. Різниця між OracleJDK та OpenJDK майже відсутня окрім ліцензії яка надає підтримку версії Java протягом певного часу та декількох бібліотек на які ліцензія GPL не розповсюджується [8]. Дата офіційного випуску 23 травня 1995 року. На 2019 рік Java один із найпопулярніших мов програмування

Програми на Java транслюються в байт-код Java, який виконується за допомогою JVM — програмою яка обробляє байт-код і передає інструкції машині як інтерпретатор.

Перевагою такого способу являється повна незалежність байт-коду, обладнання на операційної системи, що надасть змогу запустити програму на будь-якому пристрої де знаходиться відповідна JVM.

Наступною важливою перевагою являється гнучка система безпеки, в межах якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції які переходять рамки повноважень встановлені програмою будуть відразу перервані.

Ідеї які були закладені в концепцію і різні реалізації середовища JVM, надихнули багатьох ентузіастів на розширення кількості мов програмування які могли б бути використані для створення програм, виконуваних на віртуальній машині. Так ці ідеї занесені в специфікацію інфраструктури CLI, закладену в основу платформи .Net компанією Microsoft.

3.2 Технологія Spring

Spring Framework забезпечує комплексну модель розробки і конфігурації для сучасних додатків на Java на будь-яких платформах. Ключовим елементом Spring являється підтримка інфраструктури на рівні програми тому розробники можуть зосередитись на самій розробці без лишніх налаштувань конфігурацій та середовища виконання.

Основними властивостями Spring являються:

- ін’єкція залежностей;
- інверсія контролю;
- аспектно-орієнтоване програмування;
- декларативне управління транзакціями;
- створення Spring MVC web-сервісів і RESTful web-сервісів;
- початкова підримка JDBC, JPA, JMS;
- забезпечення модуля для тестів який надає підтримку для легкого написання тестів на функціонал програми.

За допомогою одного із супроводжуючих проектів Spring а саме Spring Boot вдалося легко підняти серверну частину та за допомогою самого фреймворку автоматично налаштувати конфігурації для інтерфейсу користувача та зв’язку з базою даних.

3.3 Технологія Hibernate

Hibernate — це одна з найбільш популярних реалізацій ORM-моделі на Java. Об’єктно-реляційна модель описує відносини між програмними об’єктами і записами в базі даних.

Hibernate автоматизує створення каркасу бази даних на основі описаних ORM-моделях та полегшує роботу з базою даних. Завдяки цьому нам не довелося писати SQL-скрипт для створення бази даних, а за допомогою Hibernate та Spring нам вдалося створити вбудовану базу даних яка створюється при кожному запуску програми та видаляється в кінці роботи програми.

Основними перевагами Hibernate являється:

- Hibernate набагато краще ніж звичайний JDBC;
- можливість зв’язувати об’єкти в програмі з таблицями в базі даних;
- багато шарова архітектура;
- незалежність від бази даних;
- можливість кешування даних.

Перетворення об'єктів Java на таблиці бази даних і навпаки називається об'єктно-реляційним відображенням (ORM). Java Persistence API (JPA) - це один з можливих підходів до ORM. За допомогою JPA розробник може відображати, зберігати, оновлювати і отримувати дані з реляційних баз даних на об'єкти Java і навпаки. JPA можна використовувати в додатках Java-EE і Java-SE.

JPA дозволяє розробнику працювати безпосередньо з об'єктами, а не з операторами SQL. Реалізація JPA, як правило, називається провайдером персистенції.

Перетворення між об'єктами Java і таблицями баз даних визначається за допомогою метаданих персистенції. Провайдер JPA буде використовувати інформацію метаданих персистенції для виконання правильних операцій бази даних.[9]

3.4 Мова програмування JavaScript

JavaScript — це динамічна мова програмування. Вона підтримує об’єктно-орієнтований, імперативний та функціональні стилі написання. JavaScript являється

реалізацією мови програмування ECMAScript але вперше був відомий як LiveScript. На хвилі популярності Java компанія Netscape яка була власником змінила назву на JavaScript.

Перша специфікація JavaScript включала в собі такі положення:

- JavaScript є легкою, інтерпретованою мовою програмування;
- призначений для створення мережевих орієнтованих додатків;
- доповнює та інтегрується з Java;
- доповнює і інтегрується з HTML;
- відкрита і крос-платформна.

JavaScript займає найбільшу нішу в своєму використанні у браузерах як мова сценаріїв для надання інтерактивності веб-сторінкам. JavaScript надає змогу відійти від статичних веб-сторінок до динамічних, та створити перевірку даних на стороні клієнта.

Переваги JavaScript:

- менша взаємодія з сервером;
- негайний зворотний зв'язок з відвідувачами;
- підвищена інтерактивність;
- більш чудові інтерфейси.

Обмеження JavaScript:

- на стороні клієнта не дозволяється читати або писати файли;
- не можна використовувати для мережевих програм;
- відсутні багатопоточні або багатопроцесорні можливості.

3.5 Технологія Angular 7

Angular — це платформа для побудови клієнтських додатків використовуючи HTML та TypeScript. Сама платформа Angular написана на TypeScript та включає в

себе основні та додаткові функціональності як набір TypeScript бібліотек які можна імпортувати у власний проект.

NgModules забезпечує компіляцію компонентів та збирати відповідний код у функціональні набори. Кожний додаток завжди повинен мати кореневий модуль який завантажує початкову програму та включає в себе інші модулі які описані в програмі.

Основними перевагами Angular являється:

- архітектура на основі компонентів, що забезпечує більш високу якість коду;
- TypeScript: кращий інструментарій, більш чистий код і більш висока масштабованість;
- RxJS: ефективне, асинхронне програмування;
- висока продуктивність;
- довгострокова підтримка Google;
- безпосереднє оновлення за допомогою Angular CLI;
- потужна екосистема.

3.6 Бібліотека edu.stanford.protege

Для аналізу файлу онтології була використана бібліотека edu.stanford.protege. За допомогою даної бібліотеки з легкістю можна проаналізувати файл та витягнуту певну інформацію, наприклад :

- назви класів;
- типи даних;
- URI онтології;
- екземпляри класів.

3.7 Середовище розробки

Середовище розробки даного продукту було обрано Intellij IDEA для серверної частини та WebStorm для клієнтського інтерфейсу. [10]

IntelliJ IDEA - інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, зокрема Java, PHP, Python, розроблена компанією JetBrains.

Перша версія з'явилася в січні 2001 року і швидко набула популярності як перше середовище для Java з широким набором інтегрованих інструментів для рефакторингу. Дизайн середовища орієнтований на продуктивність роботи програмістів, дозволяючи сконцентруватися на функціональних завданнях, в той час як IntelliJ IDEA бере на себе виконання рутинних операцій.

WebStorm - інтегроване середовище розробки на JavaScript, CSS & HTML від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA.

WebStorm забезпечує автодоповнення, аналіз коду на льоту, навігацію по коду, рефакторинг, налагодження, і інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами. Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладено Javascript) - тобто в таких конструкціях підтримується коректний рефакторинг.

Серед основних функцій які забезпечує JetBrains свої додатки це:

- підтримка інструментів для збірки проекту;
- ведення контролю версій;
- знаходження дубльованого коду;
- швидкий пошук по проекту;
- дебагер;
- допомога в оптимізації коду.

3.8 Висновки до розділу

У даному розділі було розглянуто інструменти та технології, які були необхідні для розробки додатку міграції даних з реляційної бази даних до онтології .owl формату. Розроблений додаток був розділений на дві частини: серверну та інтерфейс користувача. Серверна частина була розроблена на об’єктно-орієнтованій мові програмування Java з використанням фреймворків Spring та Hibernate. Для розробки інтерфейсу користувача було обрано мову програмування JavaScript та фреймворку Angular 7. Були обрані найпопулярніші середовища розробки IntellijIDEA та WebStorm які розробляє компанія JetBrains. Для аналізу файлу онтології було обрано бібліотеку edu.stanford.protege, яка справилась з завданнями які на неї були покладені.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Враховуючи що результатом роботи буде веб-програма було обрано одну з популярних мов програмування для написання серверної частини Java та мову програмування JavaScript для написання клієнтського інтерфейсу. Так як це веб-додаток в його основу було закладено шаблон проектування MVC. Навколо даного шаблону будувалась сама архітектура проекту включаючи структуру реляційної бази даних.

4.1 Шаблон MVC

MVC — це шаблон проектування веб-програм, який включає в себе декілька інших шаблонів проектування. При використанні MVC моделі даних додатку для побудови архітектури програми, інтерфейс користувача і взаємодія з користувачем буде розділена на три незалежних один від одного компонента, завдяки чому модифікація одного буде завдавати мінімального впливу на інші модулі або не впливати зовсім.

Основна ідея застосування MVC складається в розділенні даних і бізнес-логіки від інтерфейсу. За рахунок такого розділення збільшується можливість повторного використання програмного коду і спрощення його підтримки та читання у наступних ітерація розробки.

Концепція MVC розділяє дані, представлення і обробку дій користувача на компоненти:

- Модель (Model) – представляє об'єктну модель деякої предметної області, включає в себе дані і методи роботи з цими даними, реагує на

запити які приходять з контролера, повертаючи дані або змінюючи свій стан, при цьому модель не містить в собі інформації, як дані можна відобразити, а також не "спілкується" з користувачем безпосередньо.

- Подання (View) - відповідає за відображення інформації (візуалізацію), одні і ті ж дані можуть представлятися різними способами, наприклад, колекцію об'єктів за допомогою різних представлень можна уявити як в табличному вигляді, так і списком.
- Контролер (Controller) - забезпечує зв'язок між користувачем і системою, використовує модель і уявлення для реалізації необхідної реакції на дії користувача, як правило, на рівні контролера здійснюється фільтрація отриманих даних і авторизація (перевіряються права користувача на виконання дій або отримання інформації).

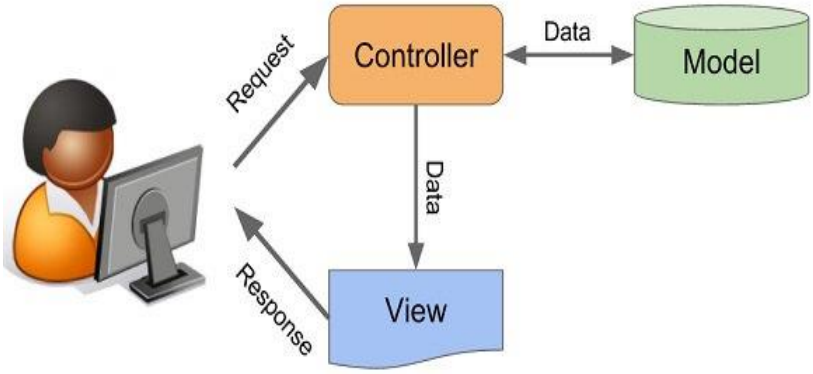


Рисунок 4.1 – Графічне представлення роботи шаблону MVC

Основними перевагами даного шаблону проектування є:

- синхронна розробка;
- висока згуртованість;
- низька прив’язка до інших контролерів та моделей;
- легкість модифікації;
- кілька переглядів для моделі.

4.2 Архітектура проекту

Так як в основі архітектури проекту лежить шаблон проектування MVC в подальшій побудові проекту використовувався архітектурний стиль REST.

REST — це стиль архітектури програмного забезпечення для розподілених систем. Зв'язок між сервером та клієнтом здійснюється повідомленнями по HTTP протоколу. Основини форматами передачі даних являється JSON та XML.

Архітектура проекту була розділена на 3 шари, а саме:

1. Шар зв'язку з реляційною базою даних
2. Шар бізнес логіки
3. Шар представлення інтерфейсу користувача

На рисунку 4.2 відображається зв’язок між трьома шарами, архітектура була побудована так щоб користувач не зміг напряму вплинути на дані в реляційній базі даних.

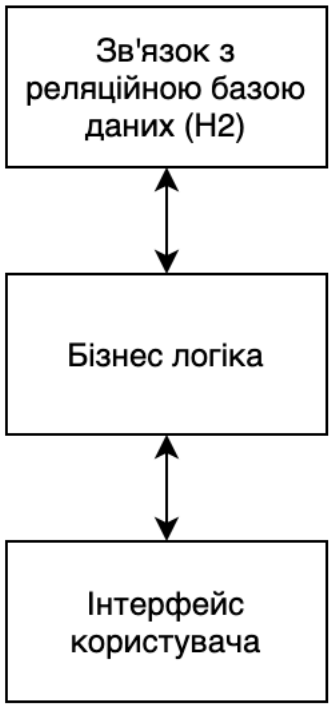


Рисунок 4.2 – Зв'язок між трьома шарами архітектури

Під час роботи користувача з програмою основними об’єктами які оброблюються це файл онтології, БД користувача та SQL-запит. В кінці ми отримаємо оновлений файл онтології в якому будуть перенесені дані з БД користувача. Рисунок 4.3.

Рисунок 4.3 – Процес міграції

4.3 Структура БД та спосіб використання

База даних використовувалась для збереження завантажених файлів онтології та параметрів для підключення до бази даних. Для даного проекту була обрана реляційна база даних H2. Так як при проектуванні системи було враховано простоту запуску програмного додатку було вирішено використовувати вбудовану базу даних яка буде створюватись та видалятися з пам’яті при кожному запуску та завершення роботи сервера. БД H2 виявилась найпопулярнішою та найменше вимагала ресурсів для своєї роботи.

Всього база даних складається з 5 таблиць які можна розглянути на рисунку 4.2

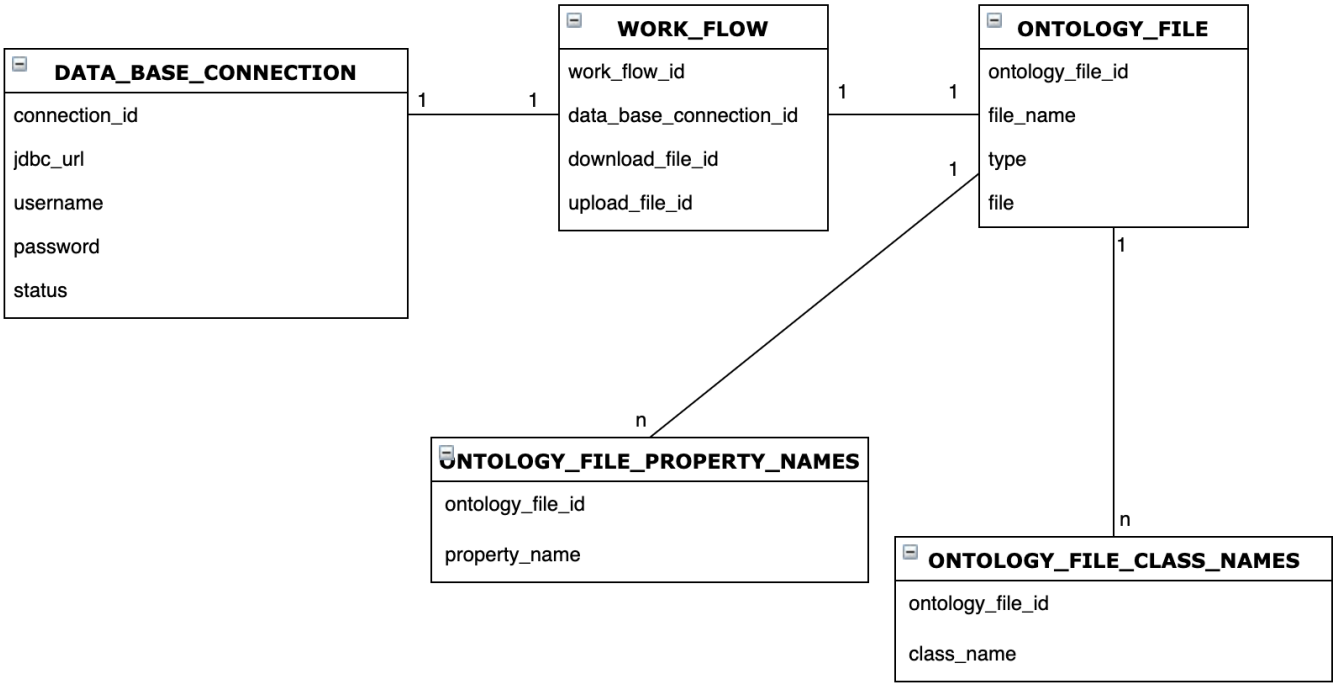


Рисунок 4.2 – Структура бази даних

У даній базі знаходяться наступні таблиці:

- DATA_BASE_CONNECTION – таблиця в якій міститься параметри для підключення до бази даних;
- WORK_FLOW – збереження інформації про процес який відбувається в даний етап, де знаходяться поточне підключення до бази даних та завантажений файл онтології;
- ONTOLOGY_FILE – інформація про завантажений файл онтології;
- ONTOLOGY_FILE_CLASS_NAMES – інформація про класи які находяться в онтології;
- ONTOLOGY_FILE_PROPERTY_NAMES – інформація про типи даних які находяться в онтології.

За допомогою Hibernate для БД створено моделі для відображення даних. Для таблиць DATA_BASE_CONNECTION та WORK_FLOW було відповідно створено класи DataBaseConnection та WorkFlow. Для таблиць ONTOLOGY_FILE, ONTOLOGY_FILE_PROPERTY_NAMES та ONTOLOGY_FILE_CLASS_NAMES створена одна модель як містить в собі всі дані з таблиці ONTOLOGY_FILE а дві

інші таблиці містяться в ній як список назв класів та властивостей, які завжди синхронізуються по ключу файлу онтології.

За допомогою Spring для кожної моделі було створено репозиторій, який автоматично надає методи для роботи з таблицями в БД без написання SQL-запитів. Дану ієрархію можна побачити на рисунку 4.3.

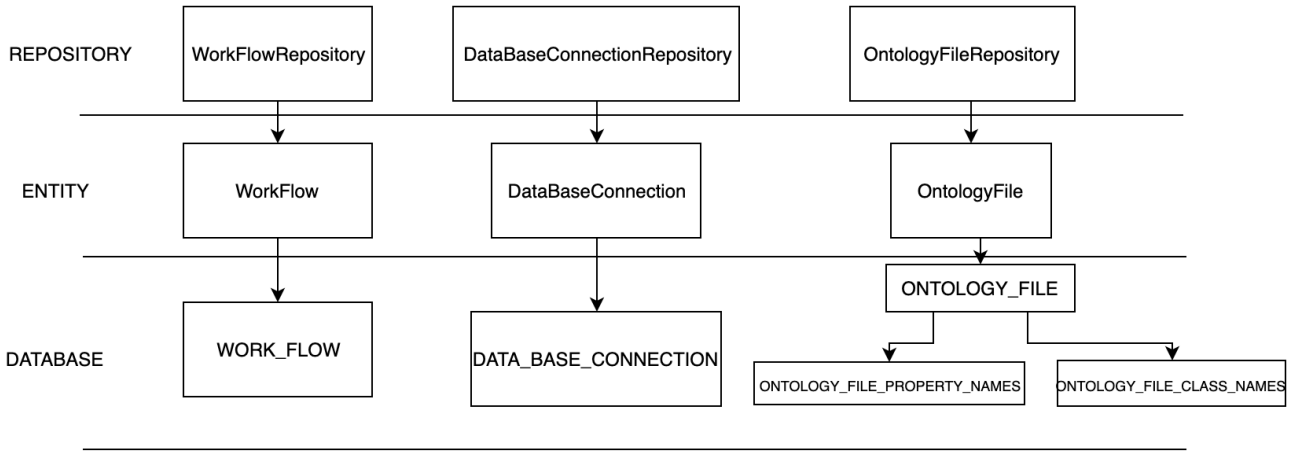


Рисунок 4.3 – Архітектура роботи з БД

Далі кожний репозиторій можна використовувати будуючи бізнес-логіку, при цьому потрібно лише оголосити даний репозиторій в потрібному класі бізнес-логіки.

4.4 Бізнес-логіка

Бізнес логіка в проекті представлена за допомогою чотирьох сервісів:

- DataBaseService – відповідає за підключення до БД користувача;
- FileService – відповідає за аналіз файлу онтології;
- OntologyService – відповідає за створення та редагування файлу онтології, валідації SQL-запиту;
- WorkFlowService – відповідає за те щоб тримати всю інформацію про поточний процес (підключення до БД, завантажений файл онтології).

Основну функціональність сервісів та репозиторії які вони використовують можна розглянути на рисунку 4.4.

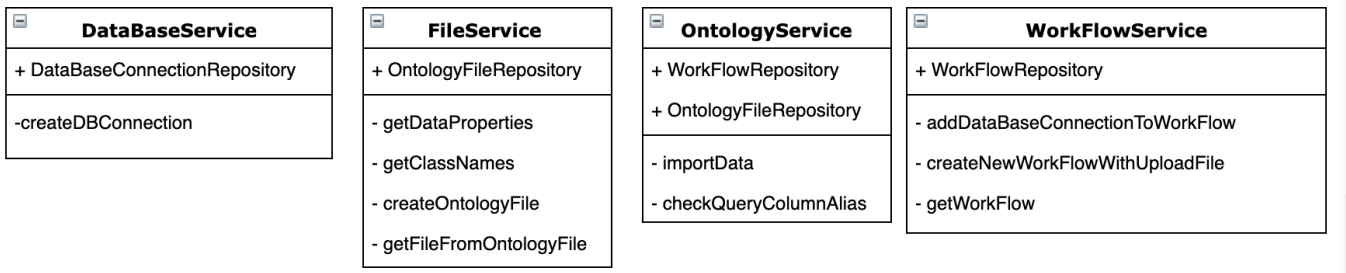


Рисунок 4.4 – Функціональність сервісів

Для аналізу даних у FileService створено два методи: getDataProperties та getClassNames. Алгоритм обох методів дуже подібний але має деяку відмінність при фільтрації даних щоб відрізняти клас від властивості класу. Алгоритм аналізу представлений на рисунку 4.5.

Рисунок 4.5 – Алгоритм аналізу отології

Для валідації SQL-запиту були поставлені наступні обмеження:

1. Коректність самого SQL-запиту.
2. Кожній колонці повинна мати назву, яка відповідає одному із властивостей класу онтології.
3. Не повинно бути дві однакові назви колонки.
4. Одна із колонок повинна містити назву «individualNameId».

Для коректного переносу даних потрібно детально проаналізувати які типи даних використовуються реляційною базою та онтологією.

Відповідно до найпопулярніших типів які використовуються в базі даних було у відповідь поставлені типи даних які пропонуються у онтології:

- boolean - http://www.w3.org/2001/XMLSchema#boolean,
- byte - http://www.w3.org/2001/XMLSchema#byte,
- dateTime - http://www.w3.org/2001/XMLSchema#dateTime,
- dateTimeStamp, http://www.w3.org/2001/XMLSchema#dateTimeStamp,
- decimal - http://www.w3.org/2001/XMLSchema#decimal,
- double - http://www.w3.org/2001/XMLSchema#double,
- float - http://www.w3.org/2001/XMLSchema#float,
- int - http://www.w3.org/2001/XMLSchema#int,
- integer - http://www.w3.org/2001/XMLSchema#integer,
- short - http://www.w3.org/2001/XMLSchema#short,
- string - http://www.w3.org/2001/XMLSchema#string.

4.5 Зв’язок з інтерфейсом користувача

Зв’зок серверної частини на інтерфейсу користувача налаштований через контролери, коли запит приходить на сервер його спочатку оброблює

DispatcerServlet а далі на основі шляху та типу запиту відправляє на відповідний контролер.

Всього в розробленому додатку 5 контролерів:

- /processOntology (POST);
- /checkSQLQuery (POST);
- /download (GET);
- /addRecordsToOntology (POST);
- /enableDBConnection (POST).

Всього в додатку використовується два типи запитів POST та GET. POST використовується для того щоб передати об’ємну інформацію та зберегти її на сервері або обновити її. GET використовується для того щоб запитати інформацію у сервера та відобразити користувачу.

Загальну схему контролерів можна подивитись на рисунку 4.6

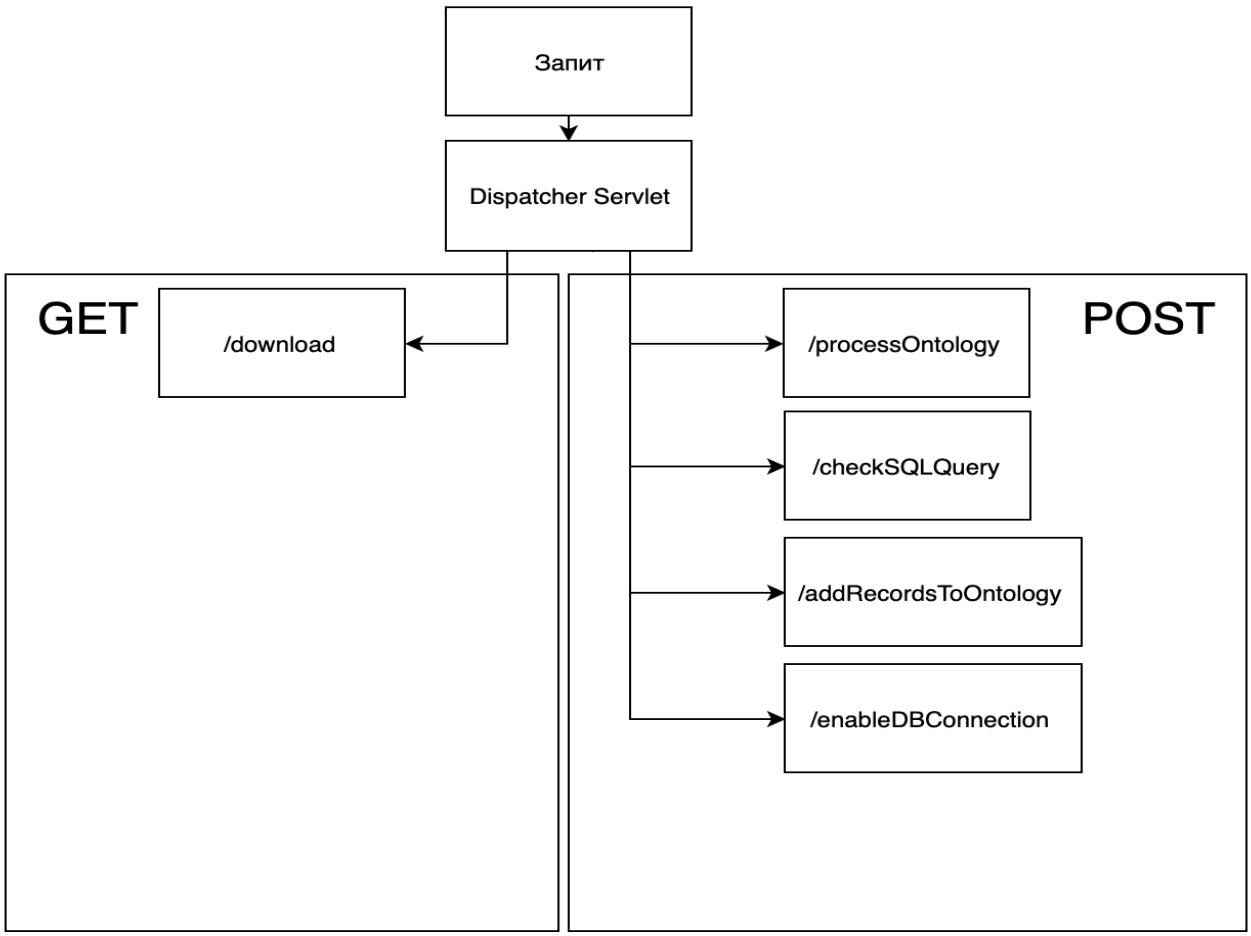


Рисунок 4.6 – Схема контролерів.

З рисунку ми можемо детальніше розглянути як розміщена контролери та їх розподілення в системі.

4.6 Висновки до розділу

В даному розділі описано особливість архітектури даного додатку. Було наведено приклад використання фреймворка Spring та Hibernate для побудови архітектури підключення до БД. Наведена структура БД в якій зберігаються дані про завантаженні файли, їх аналіз та підключення до БД користувача.

Наведено алгоритми розроблені для бізнес логіки, обмеження які використовуються для валідації SQL-запиту та принцип конвертацію типу даних з БД до онтології.

Була описана схема контролерів які використовуються для зв’язку серверної так клієнтської частини.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМОЮ

5.1 Міграція даних в файл онтології

Для роботи із системою достатньо завантажити її на веб сервер і надати користувачам доступ до нього. Для цього потрібно відкрити серверну частину через Intelij IDEA там частину клієнтського інтерфейсу через WebStormта запустити їх натиснувши кнопку «Run». Після запуску потрібно відкрити браузер та перейти за адресою http://localhost:4200/ .

Для початку роботи із системою користувачу необхідно завантажити файл із даними онтології у форматі .owl, такий файл можна отримати зберігши онтологію за допомогою будь-якого редактора онтологій, наприклад Protégé. Початкова сторінка зображена на рисунку 5.1.

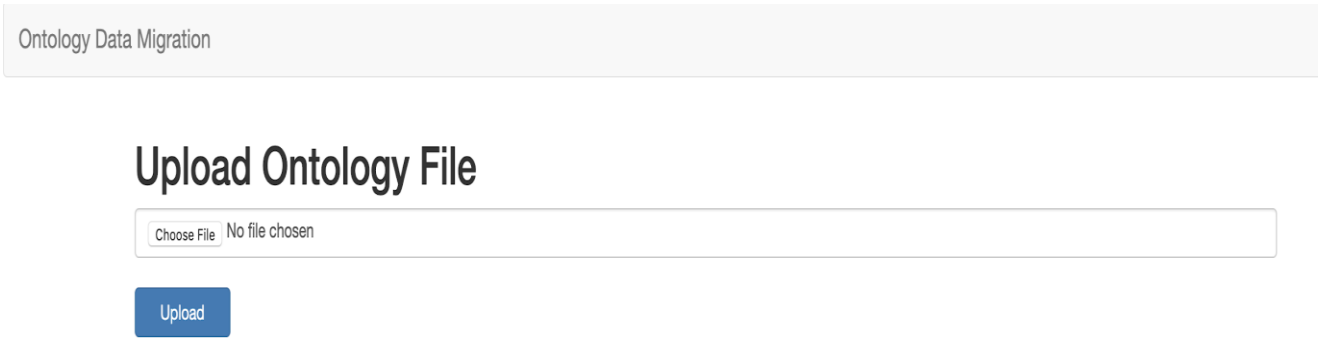


Рисунок 5.1 — Початкова сторінка

Після завантаження файлу онтології буде відображено проаналізовані класи та їхні властивості. Коли користувач перегляне чи всі дані правильно проаналізовані він

може перейти на наступний етап підключення до реляційної бази даних, натиснувши кнопку «Next». (Рисунок 5.2)

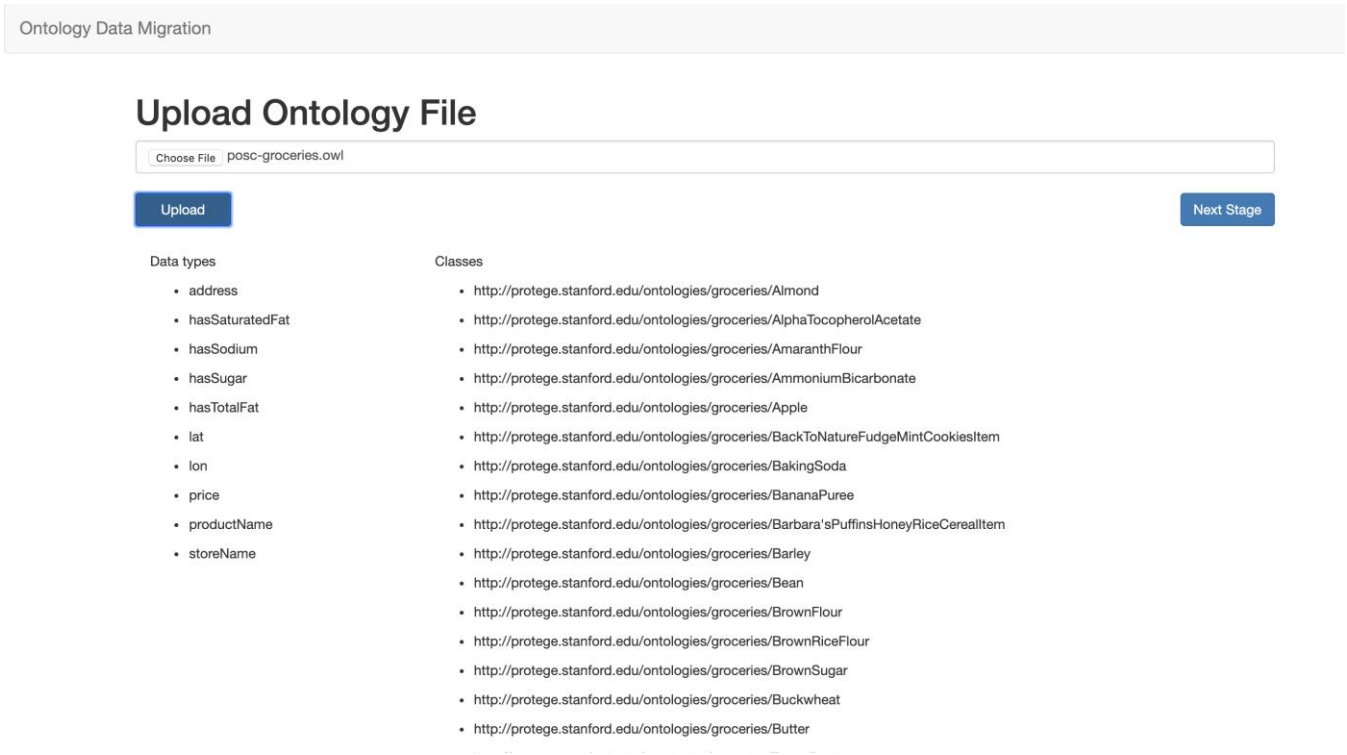


Рисунок 5.2 – Результат проаналізованого файлу онтології

Після вибору файлу онтології і його завантаження, користувач переходить на сторінку підключення до реляційної бази даних.

Підключення до реляційної бази даних може відбуватися двома шляхами, через JDBC Connection або Connection String. Для таких СКБД як H2, MySQL, Oracle, Postgres буде використовуватись JDBC Connection. Якщо буде потреба підключитися до MS SQL Server можна використати підключення за допомогою Connection String. Після натискання кнопки “Test” відбувається тестове підключення до реляційної бази даних, якщо підключення успішне з’являється кнопка “Next” (рисунок 5.3), в іншому випадку буде відображена помилка невдалого підключення.

Рисунок 5.3 — Перехід на етап написання SQL-запиту

Після натискання кнопки “Next” користувач перейде на сторінку написання тестового SQL-запиту для отримання полів які будуть перенесені в онтологію. (рисунок 5.4)

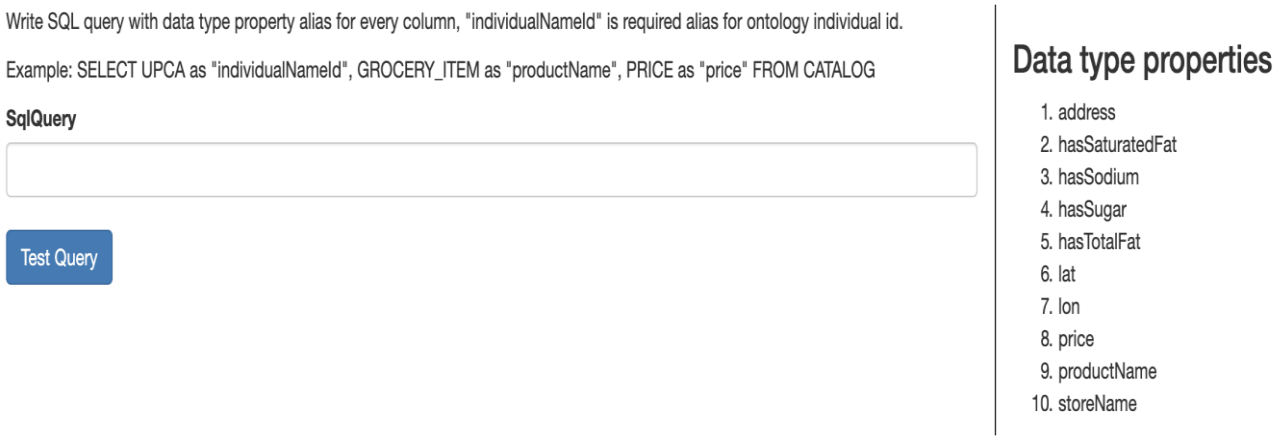


Рисунок 5.4 — Написання тестового SQL-запит

На даному етапі потрібно написати SQL-запит який буде витягувати значення для екземпляра в онтологію. З правого боку наведені типи полів, які використовуються в онтології та які можна використати при міграції даних. Далі потрібно написати сам SQL-запит в якому для кожної колонки оголосити назву, яка відповідає властивості класу з онтології. Всі колонки повинні мати ім’я однієї з властивостей класу і не повинно бути дублів назв колонок в одному запиті, обов’язково потрібно оголосити одне поле як “individualNameId” яке буде виступати ідентифікатором екземпляра класу в онтології. Якщо SQL-запит буде валідним,

користувач автоматично перейде на наступний етап. Далі користувачу потрібно вибрати один із класів який міститься в онтології (рисунок 5.5).

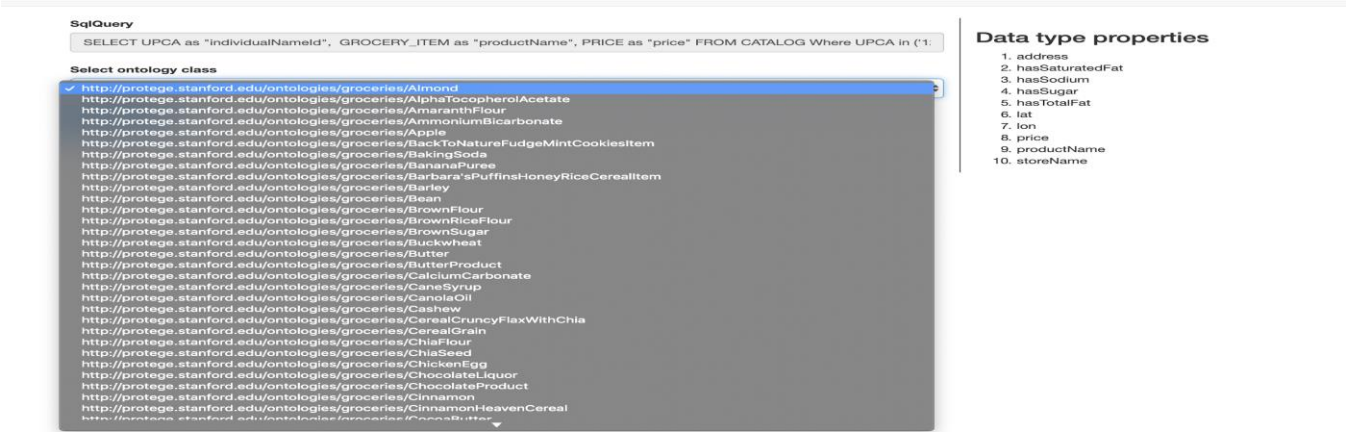


Рисунок 5.5 — Вибір класу онтології для міграції даних

Дане вікно майже аналогічне вікну зображеному на рисунку 5.4, єдиною відмінністю є випадаючий список в якому перераховані всі класи онтології з яких потрібно вибрати один для міграції даних. Для міграції даних необхідно лише натиснути кнопку “Add records”, після цього буде завантажено новий файл онтології в якому міститимуться екземпляри класу.

5.2 Перевірка коректності міграції даних

Для перевірки правильності міграції даних можна відкрити файл в текстовому редакторі і помітити його зміни. В файлі будуть помітні нові екземпляри класу, відформатовані для .owl файлу (рисунок 5.7)

```
<owl:NamedIndividual rdf:about="http://protege.stanford.edu/ontologies/groceries/346-88-3329">
        <rdf:type rdf:resource="http://protege.stanford.edu/ontologies/groceries/NutritionalInformation"/>
        <productName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Back To Nature Fudge Mint Cookies</productName>
        <price rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">3.69</price>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="http://protege.stanford.edu/ontologies/groceries/515-75-4988">
        <rdf:type rdf:resource="http://protege.stanford.edu/ontologies/groceries/NutritionalInformation"/>
        <productName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Barbara's Puffins Honey Rice Cereal</productName>
        <price rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">4.71</price>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="http://protege.stanford.edu/ontologies/groceries/127-20-5192">
        <rdf:type rdf:resource="http://protege.stanford.edu/ontologies/groceries/NutritionalInformation"/>
        <productName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Lotus Biscoff Cookies</productName>
        <price rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">4.63</price>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="http://protege.stanford.edu/ontologies/groceries/592-20-3346">
        <rdf:type rdf:resource="http://protege.stanford.edu/ontologies/groceries/NutritionalInformation"/>
        <productName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Enjoy Life Perky's Cereal Cruncy Flax with Chia</productName>
        <price rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">3.84</price>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="http://protege.stanford.edu/ontologies/groceries/214-23-4154">
        <rdf:type rdf:resource="http://protege.stanford.edu/ontologies/groceries/NutritionalInformation"/>
        <productName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Familia Swiss Muesli Mixed Cereals</productName>
        <price rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">5.08</price>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="http://protege.stanford.edu/ontologies/groceries/297-05-4265">
        <rdf:type rdf:resource="http://protege.stanford.edu/ontologies/groceries/NutritionalInformation"/>
        <productName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">General Mills Cheerios</productName>
        <price rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">3.08</price>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="http://protege.stanford.edu/ontologies/groceries/228-30-4504">
        <rdf:type rdf:resource="http://protege.stanford.edu/ontologies/groceries/NutritionalInformation"/>
        <productName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Heinz Beans with Tomato Sauce</productName>
        <price rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">5.54</price>
    </owl:NamedIndividual>
```

Рисунок 5.7 — Структура файлу онтології після міграції даних

Також можна перевірити за допомогою програм які працюють з файлами .owl формату. Наприклад, Protege. Відкривши онтологію ми помітимо, що змінилося значення поля “Individual count”, це і є екземпляри класу після міграції даних з реляційної бази даних (рисунок 5.8).

Ontology metrics:

Metrics	
Axiom	689
Logical axiom count	441
Declaration axioms count	222
Class count	195
Object property count	8
Data property count	12
Individual count	7
Annotation Property count	4
Class axioms	
SubClassOf	373
EquivalentClasses	0
DisjointClasses	27
GCI count	0
Hidden GCI Count	0
Object property axioms	
SubObjectPropertyOf	3
EquivalentObjectProperties	0

Рисунок 5.8 — Зміни в метриці файлу онтології

Для остаточної перевірки потрібно виконати SPARQL запит в самій онтології. SPARQL (рекурсивний акронім від англ. SPARQL Protocol and RDF Query Language) — мова запитів до даних, представлених по моделі RDF, а також протокол для передачі цих запитів і відповідей на них. Після виконання запиту можна переглянути результат який ми можемо спостерігати нижче(рисунок 5.9) .

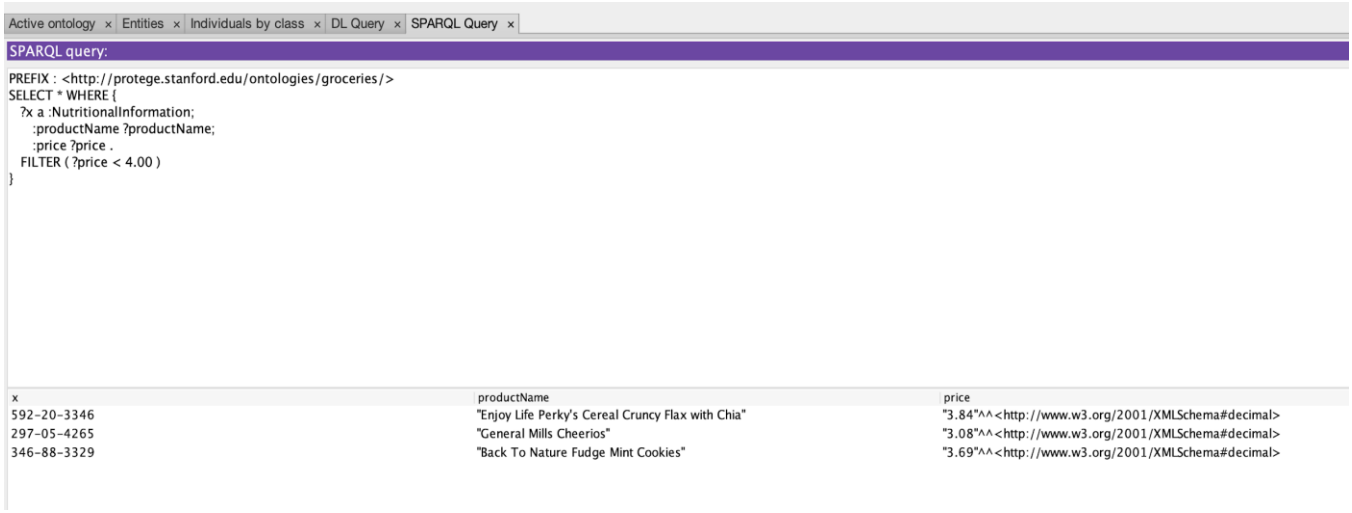


Рисунок 5.9 — Вікно виконання SPARQL запиту в Protege

5.3 Висновки до розділу

В даному розділі наведено інтерфейс користувача розробленого додатку. В процесі було завантажено файл онтології, встановлене підключення до реляційної бази даних та здійснена міграція даних до файлу онтології .owl формату.

Після міграції даних результат був перевірений у редакторі Protégé. Було продемонстровано результат та форматування самого файлу який не порушує основного форматування файлу.

ВИСНОВКИ

Під час виконання дипломної роботи було розглянуто існуючі підходи до обміну даними між онтологією та БД. Було обрано підхід з використанням однієї концептуальноюї моделі БД і онтології на основі OBDB (Ontologies Based Databases). Де реляційна БД використовується для зберігання даних, представлених в онтології.

В результаті розробки було створено програмний додаток, який дозволяє перенести дані із реляційної бази даних до онтології .owl формату. Автоматизувати процес заповнення даними онтології для зменшення використаного часу оператора.

Програмний засіб розроблений для комп'ютерів з будь-якою операційною системою, так як це WEB-додаток використати його може кожний в кого є браузер.

Вхідні дані: файл онтології з розширенням *.owl, параметри для підключення реляційної бази даних, SQL-запит в реляційну базу даних.

Вихідні дані: файл онтології з розширенням *.owl.

Основні завдання які були вирішені при розробці програмного продукту являються:

- завантаження файлу онтології з комп'ютера користувача;
- аналіз файлу онтології;
- підключення до реляційної бази даних;
- валідація підключення до реляційної бази даних
- написання SQL-запиту для отримання даних з реляційної бази даних
- валідація коректності SQL-запиту
- запис екземплярів класів в онтологію;
- завантаження оновленого файлу онтології.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лапшин, В.А. Онтологии в компьютерных системах [Текст] / В.А. Лапшин. — М.: Научный мир, 2010. — 222 с.

2. Добров Б. В., Иванов В.В., Лукашевич Н.В., Соловьев В.Д. Онтологии и тезаурусы: модели, инструменты, приложения. — М.: Бином. Лаборатория знаний, 2009. — 173 с.

3. What is an Ontology? [Електронний ресурс] – Режим доступу до ресурсу: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.

4. OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition) [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/TR/owl2-rdf-based-semantics/.

5. Triples in RDF/XML [Електронний ресурс] – Режим доступу до ресурсу: https://www.iro.umontreal.ca/~lapalme/ForestInsteadOfTheTrees/HTML/ch07s01.html.

6. Brockmans S, Haase P, Hitzler P A metamodel and uml profile for rule-extended owl dl ontologies. In: Sure JDY (ed) The semantic web: research and applications: / S. Brockmans, P. Haase, P. Hitzler – 3rd European semantic web conference, ESWC 2006, vol 4011, pp 303–316

7. Carmen Martinez-Cruz M. Amparo Vila (2012) Ontologies versus relational databases: are they so different? A comparison / Carmen Martinez-Cruz, Ignacio J. Blanco. – Artificial Intelligence Review, June 2011 10.1007/s10462-011-9251-9

8. Шилдт Г. Java. Полное руковдство / Герберт Шилдт., 2018

9. Vogel L. Java persistence API [Електронний ресурс] / L. Vogel, S. Scholz – 2019. – Режим доступу до ресурсу: https://www.vogella.com/tutorials/JavaPersistenceAPI/article.html

10. JetBrains Corporate Owerview [Електронний ресурс]. – 2018. – Режим доступу до ресурсу:

https://resources.jetbrains.com/storage/products/jetbrains/docs/jetbrains_corporate_overview.pdf.

Додаток 1

Інструментальні засоби міграції даних із реляційної бази даних до онтології .owl формату

Специфікація

УКР.НТУУ“КПІ”.ТР5168_19Б

Аркушів 2

2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТР5168_19Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ».ТР5168_19Б 12-1	Текст програмного модулю	
УКР.НТУУ«КПІ».ТР5168_19Б 13-1	Опис програми	

Додаток 2

Інструментальні засоби міграції даних із реляційної бази даних до онтології .owl формату

Текст програмного модулю

УКР.НТУУ“КПІ”.ТР5168_19Б 12-1

Аркушів 5

2019

```
package com.univerity.ontology.entity;//оголошення директорії класу

import javax.persistence.*;//імопрт бібліотек
import java.util.ArrayList;
import java.util.List;
//Створення зв’язку класом та таблицею в БД
@Entity
@Table(name = "ontology_file") //назва табилці
Модуль файлу онтології
public class OntologyFile {
    @Id //анотація для позначення змінної як ід об’єкта
    @GeneratedValue
    @Column(name = "ontology_file_id")//назва колонки в таблиці
    private Long ontologyFileId;
    @Column(name = "file_name")
    private String fileName;
    @Column(name = "type")
    private String type;
    @Lob//позначення типу змінної як файл
    @Column(name = "file")
    private byte[] file;
    @ElementCollection//по’єднання елементів іншої таблиці по ід даного об’єкту в
список
    @Column(name ="properties")
    private List<String> dataTypeProperties = new ArrayList<>();
    @ElementCollection
    @Column(name ="classes")
    private List<String> classNames = new ArrayList<>();
```

```
    public Long getOntologyFileId() {//метод для діставання змінної
        return ontologyFileId;
    }

    public void setOntologyFileId(Long ontologyFileId) { //метод для встановлення значення змінній
        this.ontologyFileId = ontologyFileId;
    }

    public String getFileName() {
        return fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public byte[] getFile() {
        return file;
    }

    public void setFile(byte[] file) {
```

```
        this.file = file;
    }

    public List<String> getDataTypeProperties() {
        return dataTypeProperties;
    }

    public void setDataTypeProperties(List<String> dataTypeProperties) {
        this.dataTypeProperties = dataTypeProperties;
    }

    public List<String> getClassNames() {
        return classNames;
    }

    public void setClassNames(List<String> classNames) {
        this.classNames = classNames;
    }
}
package com.univerity.ontology.entity;

import javax.persistence.*;

@Entity
@Table(name = "work_flow")
//Модуль поточного процесу
public class WorkFlow {
    @Id
    @GeneratedValue
    @Column(name = "work_flow_id")
```

```
    private Long workFlowId;
    @OneToOne // з’єднання з іншою таблицею один до одного
    @JoinColumn(name = "data_base_connection_id") // з’єднання по вказаному полю яке являється ключем
    private DataBaseConnection dataBaseConnection;
    @OneToOne
    @JoinColumn(name = "download_file_id")
    private OntologyFile downloadFile;
    @OneToOne
    @JoinColumn(name = "upload_file_id")
    private OntologyFile uploadFile;

    public Long getWorkFlowId() {
        return workFlowId;
    }

    public void setWorkFlowId(Long workFlowId) {
        this.workFlowId = workFlowId;
    }

    public DataBaseConnection getDataBaseConnection() {
        return dataBaseConnection;
    }

    public void setDataBaseConnection(DataBaseConnection dataBaseConnection) {
        this.dataBaseConnection = dataBaseConnection;
    }

    public OntologyFile getDownloadFile() {
        return downloadFile;
```

```
    }

    public void setDownloadFile(OntologyFile downloadFile) {
        this.downloadFile = downloadFile;
    }

    public OntologyFile getUploadFile() {
        return uploadFile;
    }

    public void setUploadFile(OntologyFile uploadFile) {
        this.uploadFile = uploadFile;
    }
}
```

Додаток З

Інструментальні засоби міграції даних із реляційної бази даних до онтології .owl формату

Опис програмного модулю

УКР.НТУУ“КПІ”.ТР5168_19Б 13-1

Аркушів 5

2019

АНОТАЦІЯ

Метою роботи було створення додатку для міграції даних із реляційної бази даних до онтології .owl формату. Додаток надає можливість в короткий час здійснити міграцію даних із реляційної бази даних до онтології .owl формату з максимально точною відповідністю типів даних БД та онтології. Програма забезпечує можливість витягнути дані з БД використовуючи SQL-запит який задовільняє потреби користувача.

ЗМІСТ

1 ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

Даний програмний модуль розроблено у середовищі WebStrom та IntelliJ IDEA, використовуючи мову програмування Java та JavaScript, фреймворки Spring, Hibernate, Angular7 та деякі додаткові бібліотеки.

Програма призначена для міграції даних із реляційної бази даних до онтології .owl формату.

1.1. Опис логічної структури

Було розроблено веб-додаток, основною задачею якого є міграція даних із реляційної бази даних до онтології .owl формату. Основними проблемами які розв’язує дана робота є автоматизоване перенесення великої кількості даних із БД до файлу онтології, звідси постає проблема в конвертації типів даних БД та онтології, та правильності запису в файл онтології, щоб файл не був пошкоджений та перенесені дані лишилися у читабельному вигляді.

1.2. Вхідні та вихідні дані

Вхідними даними для системи є файл онтології у форматі .owl, параметри підключення до реляційної бази даних та SQL-запит.

Вихідними даними є оновлений файл онтології .owl формату.

2 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний модуль було протестовано в браузері Google Chrome 74.0 на персональному комп’ютері, який працює на базі процесору 2,3 GHz Intel Core i5 та має 8 Гб оперативної пам’яті. Розроблене програмне забезпечення є кросбраузереним та кросплатформенним, що дозволяє запускати його на комп’ютерах будь-якої потужності та в будь-яких сучасних браузерах.

Додаток 4

Інструментальні засоби міграції даних із реляційної бази даних до онтології .owl формату

АПРОБАЦІЯ

Міжнародна наукова інтернет-конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» м. Тернопіль, 7 травня 2019 року

УКР.НТУУ“КПІ”.ТР5168_19Б

Аркушів 4

2019


www.konferenciaonline.org.ua
Міжнародна наукова
інтернет-конференція
Інформаційне суспільство:
технологічні, економічні
та технічні аспекти становлення
(випуск 38)
Частина 1
ISSN 2522-932X
7 травня 2019 р.
Тернопіль
2019

Талах О.М.
Національний технічний університет України «Київський політехнічний інститут
імені Ігоря Сікорського», Київ
Кафедра автоматизації проектування енергетичних процесів та систем, студент
Дацюк О.А.
Національний технічний університет України «Київський політехнічний інститут
імені Ігоря Сікорського», Київ
Кафедра автоматизації проектування енергетичних процесів та систем, старший викладач

ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ПІДКЛЮЧЕННЯ РЕЛЯЦІЙНОЇ БД ПРЕДМЕТНОЇ ОБЛАСТІ ДО ОНТОЛОГІЇ ПРЕДМЕТНОЇ ОБЛАСТІ

Онтологію застосовують як формалізоване представлення знать про певну предметну область, придатну для автоматизованої обробки. Її завжди супроводжує деяка концепція цієї області інтересів. Найчастіше ця концепція виражається за допомогою визначення базових об'єктів (індивідуумів, атрибутів, процесів) і відношень між ними [1].

Переваги онтології полягають в поліпшенні взаємодії розробників та програмних агентів, уніфікації обміну даних, формалізації процесів специфікації, підвищення надійності та забезпеченні багаторазовості використання. Онтології зазвичай містять класи, екземпляри цих класів, їхні атрибути та значення цих властивостей, а також відношення між класами та екземплярами класів. Крім того, онтологія може містити певні обмеження на використання класів та їх відношень [2]. Через це робота з онтологією напряму потребує певних навичок, а для заповнення даними потрібно використати велику кількість часу. Так в онтологію досить не зручно заносити дані. Та й працювати з великими об'ємами даних в онтології не зручно. Для цього більш пристосовані реляційні бази даних (РБД). Реляційні БД сьогодні мають досить потужні механізми БД призначені для організації коректного зберігання та швидкої обробки великих об'ємів даних. Можливість заповнення онтології з бази даних на основі співвідношення стовпців таблиць бази даних та атрибутів онтології спростить та автоматизує роботу користувача.

Зв'язок між БД та онтологією може бути встановлений, якщо інформація, представлена в онтології відповідає даним БД. При цьому організація комплексної роботи онтології з базою даних проводиться за певними правилами. Є декілька підходів для сумісного використання даних онтологією і БД.

Перший підхід ґрунтується на використанні однієї концептуальної моделі для БД і онтології. В [3] запропоновано використовувати UML-діаграму для генерації схеми онтології та БД.

Другий підхід орієнтований на створенні схем баз даних з онтологій. Звичайно, в цьому випадку при записі великих об'ємів даних може бути втрачено семантичний зв'язок між даними.

Протилежним підходом є отримання онтології з реляційної бази даних. Це найбільш поширений метод, але така онтологія має обмежений набір зв'язків і залежностей між класами, що легко компенсується ручним коригуванням структури онтології [4].

Одним з різновидів першого підходу є використання БД на основі онтологій OBDB (Ontologies Based Databases). Де реляційна БД використовується для зберігання даних, представлених в онтології. В даному випадку схема БД не завжди співпадає зі схемою онтології, але робота підтримується за допомогою окремих спеціально створених комунікативних зв'язків. Такі зв'язки часто приходиться встановлювати вручну.

Ще однією альтернативною сумісного використання БД та онтології є отнологія, яка описує структуру концептуалізація моделі реляційної бази даних.

При спробі підключити існуючу реляційну БД до онтології предметної області, було обрано варіант отримання онтології з реляційної бази даних. Звичайно в даному підході також необхідно мати спеціальні засоби одночасного доступу до даних обох моделей. Але ми отримуємо можливість підключення існуючої БД до потрібної онтології.

Основні переваги платформи:

- зручний інтерфейс;
- можливість ручного налаштування відношення бази даних та онтології;
- можливість підключення будь-якої бази даних.

Дана система реалізована за допомогою Java та Angular 7. Після розгортання програми на сервері доступ до системи можна отримати з різних операційних систем використовуючи браузер та посиланням до системи.

Можливе розширення функціоналу системи: використання даних з SQL запитів, задання додаткових параметрів для перенесення даних з бази даних в онтологію, відображення схеми бази даних у веб-інтерфейсі та створення співвідношення онтології та бази даних графічними засобами.

Література:

1. Лапшин, В.А. Онтологии в компьютерных системах [Текст] / В.А. Лапшин. — М.: Научный мир, 2010. — 222 с.

2. Добров Б. В., Иванов В.В., Лукашевич Н.В., Соловьев В.Д. Онтологии и тезаурусы: модели, инструменты, приложения. — М.: Бином. Лаборатория знаний, 2009. — 173 с.

3. Brockmans S, Haase P, Hitzler P A metamodel and uml profile for rule-extended owl dl ontologies. In: Sure JDY (ed) The semantic web: research and applications: / S. Brockmans, P. Haase, P. Hitzler – 3rd European semantic web conference, ESWC 2006, vol 4011, pp 303–316

4. Carmen Martinez-Cruz M. Amparo Vila (2012) Ontologies versus relational databases: are they so different? A comparison / Carmen Martinez-Cruz, Ignacio J. Blanco. – Artificial Intelligence Review, June 2011 10.1007/s10462-011-9251-9